Computational Quality of Service in Parallel CFD *

B. Norris,[a] L. McInnes,[a] I. Veljkovic [b]

[a]Mathematics and Computer Science Division, Argonne National Laboratory,
9700 South Cass Avenue, Argonne, IL 60439-4844, *[norris,mcinnes]@mcs.anl.gov.*

[b]Department of Computer Science and Engineering, The Pennsylvania State University,
IST Building, University Park, PA 16802-6106, *veljkovi@cse.psu.edu.*

Abstract. Our computational quality-of-service infrastructure is motivated by large-scale scientific simulations based on partial differential equations, with emphasis on multimethod linear solvers in the context of parallel computational fluid dynamics. We introduce a component infrastructure that supports performance monitoring, analysis, and adaptation of important numerical kernels, such as nonlinear and linear system solvers. We define a simple, flexible interface for the implementation of adaptive nonlinear and linear solver heuristics. We also provide components for monitoring, checkpointing, and gathering of performance data, which are managed through two types of databases. The first is created and destroyed during runtime and stores performance data for code segments of interest, as well as various application-specific performance events in the currently running application instance. The second database is persistent and contains performance data from various applications and different instances of the same application. This database can also contain performance information derived through offline analysis of raw data. We describe a prototype implementation of this infrastructure and illustrate its applicability to adaptive linear solver heuristics used in a driven cavity flow simulation code.

1. INTRODUCTION

Component-based environments provide opportunities to improve the performance, numerical accuracy, and other characteristics of parallel simulations in CFD. Because component-based software engineering combines object-oriented design with the powerful features of well-defined interfaces, programming language interoperability, and dynamic composability, it helps to overcome obstacles that hamper sharing even well-designed traditional numerical libraries. Not only can applications be assembled from components selected to provide good algorithmic performance and scalability, but they can also be changed dynamically during execution to optimize desirable characteristics. We use the term *computational quality of service* (CQoS) [9, 16] to refer to the automatic selection and configuration of components for a particular computational purpose. CQoS embodies the familiar concept of quality of service in networking and

*This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Advanced Scientific Computing Research, Office of Science, U.S. Department of Energy, under Contract W-31-109-ENG-38.

the ability to specify and manage characteristics of the application in a way that adapts to the changing (computational) environment. The factors affecting performance are closely tied to a component's parallel implementation, its management of memory, the algorithms executed, the algorithmic parameters employed (e.g., the level of overlap in an additive Schwarz preconditioner), and other operational characteristics. CQoS is also concerned with functional qualities, such as the level of accuracy achieved for a particular algorithm.

This paper presents an overview of new software infrastructure for automated performance gathering and analysis of high-performance components, a key facet of our CQoS research, with emphasis on using these capabilities in parallel CFD simulations, such as flow in a driven cavity and compressible Euler flow. The remainder of this paper is organized as follows. Section 2 discusses parallel CFD applications and algorithms that motivate this infrastructure. Section 3 introduces the new framework for enabling CQoS in parallel nonlinear PDE-based applications. Section 4 illustrates the performance of a simple adaptive algorithm strategy. Section 5 concludes with a summary and discussion of future work.

2. MOTIVATING APPLICATIONS AND ALGORITHMS

Flow in a Driven Cavity. The first parallel application that motivates and validates this work is driven cavity flow, which combines lid-driven flow and buoyancy-driven flow in a two-dimensional rectangular cavity. The lid moves with a steady and spatially uniform velocity and thus sets a principal vortex and subsidiary corner vortices. The differentially heated lateral walls of the cavity induce a buoyant vortex flow, opposing the principal lid-driven vortex. We use a velocity-vorticity formulation of the Navier-Stokes and energy equations, which we discretize using a standard finite-difference scheme with a five-point stencil for each component on a uniform Cartesian mesh; see [7] for a detailed problem description.

Compressible Euler Flow. Another motivating application is PETSc-FUN3D [2], which solves the compressible and incompressible Navier-Stokes equations in parallel; the sequential model was originally developed by W. K. Anderson [1]. The code uses a finite-volume discretization with a variable-order Roe scheme on a tetrahedral, vertex-centered unstructured mesh. The variant of the code under consideration here uses the compressible Euler equations to model transonic flow over an ONERA M6 wing, a common test problem that exhibits the development of a shock on the wing surface. Initially a first-order discretization is used; but once the shock position has settled down, a second-order discretization is applied. This change in discretization affects the nature of the resulting linear systems.

Newton-Krylov Algorithms. Both applications use inexact Newton methods (see, e.g., [14]) to solve nonlinear systems of the form $f(u) = 0$. We use parallel preconditioned Krylov methods to (approximately) solve the Newton correction equation $f'(u^{\ell-1})\ \delta u^{\ell} = -f(u^{\ell-1})$, and then update the iterate via $u^{\ell} = u^{\ell-1} + \alpha \cdot \delta u^{\ell}$, where α is a scalar determined by a line search technique such that $0 < \alpha \leq 1$. We terminate the Newton iterates when the relative reduction in the residual norm falls below a specified tolerance. Our implementations use the Portable, Extensible Toolkit for Scientific computation (PETSc) [3], a suite of data structures and routines for the scalable solution of scientific applications modeled by PDEs. PETSc integrates a hierarchy of libraries that range from low-level distributed data structures for vectors and matrices through high-level linear, nonlinear, and time-stepping solvers.

Pseudo-Transient Continuation. For problems with strong nonlinearities, Newton's method

often struggles unless some form of continuation is employed. Hence, we incorporate pseudo-transient continuation [11], a globalization technique that solves a sequence of problems derived from the model $\frac{\partial u}{\partial t} = -f(u)$, namely,

$$g_\ell(u) \equiv \frac{1}{\tau^\ell}(u - u^{\ell-1}) + f(u) = 0,\ \ell = 1, 2, \ldots, \tag{1}$$

where τ^ℓ is a pseudo time step. At each iteration in time, we apply Newton's method to Equation (1). As discussed by Kelley and Keyes [11], during the initial phase of pseudo-transient algorithms, τ^ℓ remains relatively small, and the Jacobians associated with Equation (1) are well conditioned. During the second phase, the pseudo time step τ^ℓ advances to moderate values, and in the final phase τ^ℓ transitions toward infinity, so that the iterate u^ℓ approaches the root of $f(u) = 0$.

Adaptive Solvers. In both applications the linearized Newton systems become progressively more difficult to solve as the simulation advances due to the use of pseudo-transient continuation [11]. Consequently both are good candidates for the use of *adaptive* linear solvers [4,5,13], where the goal is to improve overall performance by combining more robust (but more costly) methods when needed in a particularly challenging phase of solution with faster (though less powerful) methods in other phases. Parallel adaptive solvers are designed with the goal of reducing the overall execution time of the simulation by dynamically selecting the most appropriate method to match the characteristics of the current linear system.

A key facet of developing adaptive methods is the ability to consistently collect and access both runtime and historical performance data. Our preliminary research in adaptive methods [4, 5, 13], which employed ad hoc techniques to collect, store, and analyze data, has clearly motivated the need for a framework to analyze performance and help to manage algorithmic adaptivity.

3. COMPUTATIONAL QUALITY OF SERVICE FOR PARALLEL CFD

For a given parallel fluids problem, the availability of multiple solution methods, as well as multiple configurations of the same method, presents both a challenge and an opportunity. On the one hand, an algorithm can be chosen to better match the application's requirements. On the other hand, manually selecting a method in order to achieve good performance and reliable results is often difficult or impossible. Component-based design enables us to automate, at least partially, the task of selecting and configuring algorithms based on performance models, both for purposes of initial application assembly and for runtime adaptivity.

The Common Component Architecture (CCA) specification [6] defines a component model that specifically targets high-performance scientific applications, such as parallel CFD. Briefly, CCA components are units of encapsulation that can be composed to form applications; *ports* are the entry points to a component and represent public interfaces through which components interact; *provides* ports are interfaces that a component implements, and *uses* ports are interfaces through which a component invokes methods implemented by other components. A runtime *framework* provides some standard services to all CCA components, including instantiation of components and connection of *uses* and *provides* ports. At runtime, components can be instantiated/destroyed and port connections made/broken, thereby allowing dynamic adaptivity of CCA component applications and enabling the implementation of the adaptive linear solver methods introduced above.

In this paper, we present a CCA component infrastructure that allows researchers to monitor and adapt a simulation dynamically based on two main criteria: the runtime information about performance parameters and the information extracted from metadata from previous instances (executions) of a component application. This infrastructure includes components for performance information gathering, analysis, and interactions with off-line databases. Figure 1 (a) shows a typical set of components involved in nonlinear PDE-based applications; no explicit performance monitoring or adaptive method support is shown here. Figure 1 (b) shows the same application with the new performance infrastructure components. This design makes development of adaptive algorithms easier and less error-prone by separating as much as possible unrelated concerns from the adaptive strategy itself. In contrast, because our initial adaptive linear solver implementations were tightly interleaved and accessible only through a narrow PETSc interface intended for simple user-defined monitoring functions, the software [5, 13] became difficult to understand and maintain (mixed code for multiple heuristics) and extend (e.g., when adding new adaptive heuristics). This situation motivated us to define a simple interface that is flexible enough to enable the implementation of a wide range of adaptive heuristics, with an initial focus on adaptive linear solvers. We have reproduced some of our original results using this new infrastructure, incurring only the expected minimal fixed overhead of component interactions, for example as shown in [15].

We briefly introduce the terminology used in our CQoS infrastructure. We collectively refer to performance-relevant attributes of a unit of computation, such as a component, as *performance metadata*, or just *metadata*. These attributes include algorithm or application parameters, such as problem size and physical constants; compiler optimization options; and execution information, such as hardware and operating system information. Performance metrics, also referred to as CQoS metrics, are part of the metadata, for example, execution time and convergence history of iterative methods. Ideally, for each application execution, the metadata should provide enough information to duplicate the run; in practice, not all parameters that affect the performance are known or can be obtained, but the most significant ones are usually represented in the metadata we consider. We collectively refer to such metadata as an application instance, or *experiment*.

The design of our infrastructure is guided by the following goals: (1) low overhead during the application's execution: since all the time spent in performance monitoring and analysis/adaptation is overhead, the impact on overall performance must be minimized; (2) minimal code changes to existing application components in order to encourage use of this performance infrastructure by as many CCA component developers as possible; and (3) ease of implementation of performance analysis algorithms and new adaptive strategies, to enable and encourage the development and testing of new heuristics or algorithms for multimethod components.

Within our framework we differentiate between tasks that have to be completed during runtime and tasks that are performed when the experiment is finished (because of stringent overhead constraints during an application's execution). Consequently, we have two databases that serve significantly different purposes. The first is created and destroyed during runtime and stores performance data for code segments of interest and application-specific performance events for the running experiment. The second database is persistent and contains data about various applications and experiments within one application. The second database also contains metadata derived by performance analysis of raw performance results. At the conclusion of an experiment, the persistent database is updated with the information from the runtime database.

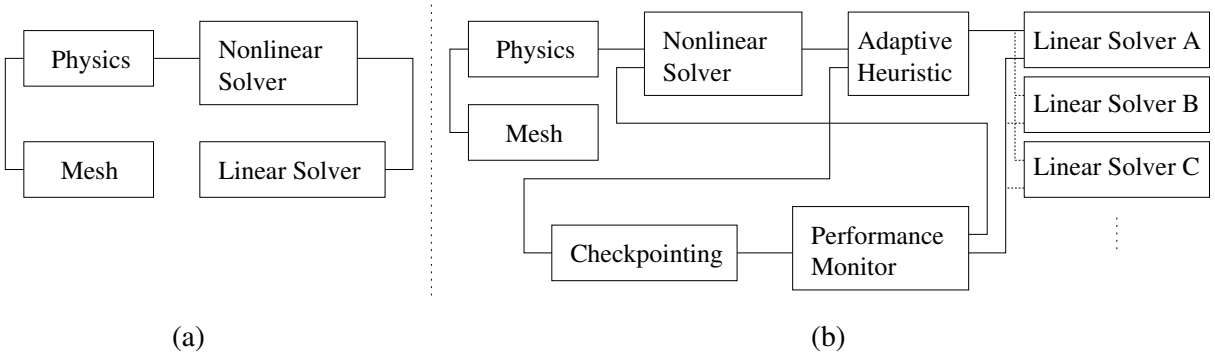

Figure 1. Some of the components and port connections in a typical PDE application: (a) in a traditional nonadaptive setting, and (b) augmented with performance monitoring and adaptive linear solver components.

Our initial implementation of this infrastructure relies on the Tuning and Analysis Utilities (TAU) toolkit [8] and the Parallel Performance Data Management Framework (PerfDMF) [10]. We now briefly describe the principal components involved in collecting and managing performance metadata and runtime adaptation.

The **Adaptive Heuristic** component implements a simple *AdaptiveAlgorithm* interface, whose single method, *adapt*, takes an argument containing application-specific metadata needed to implement a particular adaptive heuristic and to store the results. Specific implementations of the *AdaptiveContext* interface contain performance metadata used by adaptive heuristics, as well as references to the objects that provide the performance metadata contained in the context.

The **TAU Measurement** component collects runtime data from hardware counters, timing, and user-defined application-specific events. This component was provided by the developers of TAU; complete implementation details can be found in [12]. The **Performance Monitor** component monitors the application, including the selection of algorithms and parameters based on runtime performance data and stored metadata.

The **Checkpoint** component collects and stores metadata into a runtime database that can be queried efficiently during execution for the purpose of runtime performance monitoring and adaptation. When we started our implementation, the TAU profiling API could give only either callpath-based or cumulative performance information about an instrumented object (from the time execution started). Hence, we have introduced the Checkpoint component to enable us to store and retrieve data for the instrumented object during the application's execution (e.g., number of cache misses for every three calls of a particular function). The period for checkpointing can be variable; the component can also be used by any other component in the application to collect and query context-dependent and high-level performance information. For example, a linear solver component can query the checkpointing component for performance metadata of the nonlinear solver (the linear solver itself has no direct access to the nonlinear solver that invoked it). We can therefore always get the latest performance data for the given instrumented object from the database constructed during runtime.

The **Metadata Extractor** component retrieves metadata from the database at runtime. After running several experiments, analyzing the performance data, and finding a common perfor-

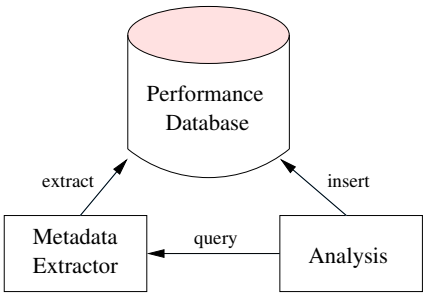

Figure 2. Components for offline query, management, and analysis of CQoS metadata.

mance behavior with some parameter values, we store data summarizing this behavior in the database. An example of derived metadata is the rate of convergence of a nonlinear or a linear solver. During runtime, these data are used in adapting our parameter and algorithm selection, and the Metadata Extractor component can retrieve compact metadata from the database efficiently.

Offline Analysis Support. The portions of the infrastructure that are not used at runtime are illustrated in Figure 2. They include a performance data extractor for retrieving data from the performance database, which is used by the offline analysis algorithm components. At present, the extractor also produces output in Matlab-like format, which is convenient for plotting some performance results; this output can be enhanced to interface with tools that provide more advanced visualization capabilities, such as an extension of ParaProf (part of the TAU suite of tools).

Many analyses can be applied offline to extract performance characteristics from the raw execution data or the results of previous analyses—in fact, facilitating the development of such analyses was one of our main motivations for developing this performance infrastructure. Initially we are focusing on simple analyses that allow us to replicate results in constructing adaptive, polyalgorithmic linear solvers from performance statistics of base linear solver experiments [4, 5, 13]. For the longer term, we plan to use this infrastructure for rapid development of new performance analyses and adaptive heuristics.

4. APPLICATION EXAMPLE

We illustrate the use of our performance infrastructure in the parallel driven cavity application briefly described in Section 2. As mentioned earlier, the use of pseudo-transient continuation affects the conditioning of the linearized Newton systems; thus, the resulting linear systems are initially well-conditioned and easy to solve, while later in the simulation they become progressively harder to solve.

In this parallel application, metadata describing the performance of the nonlinear solution, as well as each linear solution method, can be used to determine when to change or reconfigure linear solvers [5, 13]. Figure 3 shows some performance results comparing the use of a simple adaptive heuristic with the traditional single solution method approach. Even this simple automated adaptive strategy performs better than most base methods, and almost as well as the best base method (whose performance, of course, is not known a priori). Recent work on more

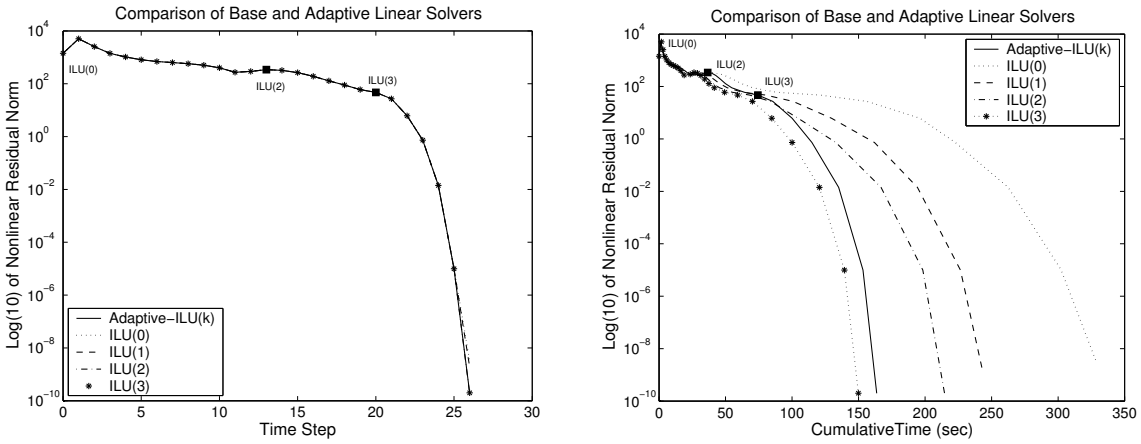

Figure 3. Comparison of single-method linear solvers and an adaptive scheme. We plot the nonlinear convergence rate (in terms of residual norm) versus both time step (left-hand graph) and cumulative execution time (right-hand graph).

sophisticated adaptive heuristics has achieved parallel performance that is consistently better than any single solution method [4].

Another use of our performance infrastructure is selection of application parameters based on performance information with the goal of maximizing performance. For example, in the application considered here, the initial CFL value is essential for determining the time step for the pseudo-transient Newton solver, which in turn affects the overall rate of convergence of the problem. One can query the database to determine the best initial CFL value from the experimental data available. Similarly, the values of other parameters that affect performance can be determined based on past performance data and offline analysis.

5. CONCLUSIONS AND FUTURE WORK

This work has introduced new infrastructure for performance analysis and adaptivity of parallel PDE-based applications, with a focus on computational fluids dynamic simulations. We are currently completing the migration of our existing adaptive heuristics to the new component infrastructure. New heuristics for adaptive method selection will also be investigated, including components for offline analyses of performance information. In addition to runtime adaptation, our performance infrastructure can potentially support initial application assembly and is being integrated with existing CCA component infrastructure that uses component performance models for automated application assembly. The accuracy of such models can be enhanced by using historical performance data to select the best initial set of components.

While our current adaptive linear solver implementations are based on PETSc solvers, the component infrastructure is not limited to one particular library. We plan to experiment with adaptive algorithms based on other libraries as well as to evaluate their effectiveness on new applications. Another topic of future research concerns defining and using CQoS metrics that reflect differences in algorithmic scalability in large-scale parallel environments. Such metrics would rely at least partly on performance models of the scalability of various algorithms.

REFERENCES

1. W. K. Anderson and D. Bonhaus. An implicit upwind algorithm for computing turbulent flows on unstructured grids. *Computers and Fluids*, 23(1):1–21, 1994.
2. W. K. Anderson, W. D. Gropp, D. K. Kaushik, D. E. Keyes, and B. F. Smith. Achieving high sustained performance in an unstructured mesh CFD application. In *Proceedings of Supercomputing 1999*. IEEE Computer Society, 1999. Gordon Bell Prize Award Paper in Special Category.
3. S. Balay, K. Buschelman, W. Gropp, D. Kaushik, M. Knepley, L. McInnes, Barry F. Smith, and H. Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 2.2.1, Argonne National Laboratory, 2004. http://www.mcs.anl.gov/petsc/ .
4. S. Bhowmick, D. Kaushik, L. McInnes, B. Norris, and P. Raghavan. Parallel adaptive solvers in compressible PETSc-FUN3D simulations. Argonne National Laboratory preprint ANL/MCS-P1279-0805, submitted to *Proc. of the 17th International Conference on Parallel CFD*, Aug 2005.
5. S. Bhowmick, L. C. McInnes, B. Norris, and P. Raghavan. The role of multi-method linear solvers in PDE-based simulations. *Lecture Notes in Computer Science, Computational Science and its Applications-ICCSA 2003*, 2667:828–839, 2003.
6. CCA Forum homepage. http://www.cca-forum.org/ , 2005.
7. T. S. Coffey, C.T. Kelley, and D.E. Keyes. Pseudo-transient continuation and differential algebraic equations. *SIAM J. Sci. Comp*, 25:553–569, 2003.
8. Department of Computer and Information Science, University of Oregon, Los Alamos National Laboratory, and Research Centre Julich, ZAM, Germany. *TAU User's Guide (Version 2.13)*, 2004.
9. P. Hovland, K. Keahey, L. C. McInnes, B. Norris, L. F. Diachin, and P. Raghavan. A quality of service approach for high-performance numerical components. In *Proceedings of Workshop on QoS in Component-Based Software Engineering, Software Technologies Conference*, Toulouse, France, 20 June 2003.
10. K. Huck, A. Malony, R. Bell, L. Li, and A. Morris. PerfDMF: Design and implementation of a parallel performance data management framework. In *Proc. International Conference on Parallel Processing (ICPP 2005)*. IEEE Computer Society, 2005.
11. C. T. Kelley and D. E. Keyes. Convergence analysis of pseudo-transient continuation. *SIAM Journal on Numerical Analysis*, 35:508–523, 1998.
12. A. Malony, S. Shende, N. Trebon, J. Ray, R. Armstrong, C. Rasmussen, and M. Sottile. Performance technology for parallel and distributed component software. *Concurrency and Computation: Practice and Experience*, 17:117–141, Feb 2005.
13. L. McInnes, B. Norris, S. Bhowmick, and P. Raghavan. Adaptive sparse linear solvers for implicit CFD using Newton-Krylov algorithms. In *Proceedings of the Second MIT Conference on Computational Fluid and Solid Mechanics*, Boston, USA, June 2003. Massachusetts Institute of Technology.
14. J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer-Verlag, 1999.
15. B. Norris, S. Balay, S. Benson, L. Freitag, P. Hovland, L. McInnes, and B. Smith. Parallel components for PDEs and optimization: Some issues and experiences. *Parallel Computing*, 28(12):1811–1831, 2002.
16. B. Norris, J. Ray, R. Armstrong, L. McInnes, Bernholdt, W. Elwasif, A. Malony, and S. Shende. Computational quality of service for scientific components. In *Proceedings of the International Symposium on Component-Based Software Engineering, (CBSE7), Edinburgh, Scotland*, 2004.